

What Is PowerShell, and Why Do You Need It?

HERE ARE THE TOPICS COVERED IN THIS CHAPTER:

► WHY POWERSHELL?	2
Overview of PowerShell	3
The Power Behind PowerShell	5
What About the Learning Curve?	6
► WHAT'S NEW IN POWERSHELL 2.0?	7
PowerShell in the Enterprise	8
PowerShell with a GUI	10
► POWERSHELL HAS SOMETHING FOR EVERYONE	13
What's in It for IT Professionals?	14
What's in It for Developers?	15

IT professionals have been looking for ways to automate and perform tasks in a consistent manner for years. There have been many techniques and technologies — from simple batch files to third-party tools — to accomplish the tasks. Some IT professionals have gone the extra step and learned developer languages, such as Visual Basic or JavaScript, to give their scripts more power.

A majority of these tools were not integrated into the Microsoft environment. More importantly, the documentation for these tools to accomplish common administrative tasks was not readily available. As part of its effort over the years to improve the scripting environment, Microsoft developed PowerShell to overcome the challenges of previous scripting languages.

PowerShell provides a common language you can use throughout your Microsoft infrastructure. You will spend less time on manual repetitive tasks by scripting these tasks with PowerShell. PowerShell is used in a number of scenarios, including system administration and software development. PowerShell is ideal for remote management, reporting, automation, and administration.

This book focuses on learning this powerful scripting language with real-world examples and ways to perform common, everyday tasks. Tasks such as backing up servers, maintaining web servers, analyzing your environment, and many more can benefit from PowerShell. Step-by-step instructions in the chapters that follow show you how you can make PowerShell work for you.

The book is divided into two sections. In the first few chapters, you will build the foundation of your PowerShell knowledge. You will learn the basics of a building block known as a *cmdlet* (pronounced “command-let”) and how to read script. The second section of the book focuses on administrative tasks you can perform in Windows Server 2008 R2. Although the book is geared to working on a Windows Server 2008 R2 setup, the foundational knowledge provided in the book allows you to leverage PowerShell regardless of the target Windows operating system. The goal is to demystify PowerShell for you so you can use it in your day-to-day tasks.

This chapter gives an overview of PowerShell and why it is important.

Why PowerShell?

If you have been working in a Microsoft environment for the past few years, you may have seen or heard about PowerShell. You may even remember its original code name, Monad. It may have been discounted as “yet another scripting language” and put aside to look at later. You may have even thought, why reinvent the wheel?

In other words, your environment was running smoothly, you were busy, and you had no time to learn the language. You may have decided to wait to see whether there would be a version 2 and whether Microsoft was really serious about this language. Well, here we are with version 2, and PowerShell is getting better than ever. Microsoft and communities such as <http://powershellcommunity.org/> are creating native PowerShell commands and providers as well as the documentation for scripts to make your everyday work with PowerShell even easier. So, you are not in this alone. The community is growing and vibrant!

The initial project Monad debuted in June 2005. In April 2006, Microsoft announced that Monad's name would be PowerShell, and PowerShell Release Candidate 1 was released. PowerShell 1.0 was released in November 2006. It was well received in the community, and with its integration into the Windows environment, this became a new language for administrators to work with. In 2009, version 2 of PowerShell was released and built into Windows 7 and Windows Server 2008 R2. PowerShell 2.0 is also available for free download for systems newer than Windows XP SP3. Chapter 2 discusses how to install the tools on older, supported operating systems.

Overview of PowerShell

What is PowerShell?

- ▶ PowerShell is an extensible automation engine from Microsoft.
- ▶ PowerShell is a command-line shell and task-based scripting technology that provides you with enhanced remote management and automation of system administration tasks.

PowerShell can look like Figure 1.1, and it can look like Figure 1.2.

What can it do?

PowerShell enables you to perform via scripts virtually any task you can do in the GUI for your local or remote Windows operating systems and your computers. With PowerShell, you can script and automate your day-to-day administrative tasks.

- ▶ Do you need to get a list of all the computers on the network and create a report on the service pack level for each operating system?
- ▶ Do you need to check to make sure that all the users in the domain are complying with the corporate password policy?
- ▶ Do you need to start a service on 500 computers?
- ▶ Do you need to add 100 user accounts to your domain?

- Do you need to collect all the critical and error events from the event logs of all your servers?

FIGURE 1.1 This is PowerShell.

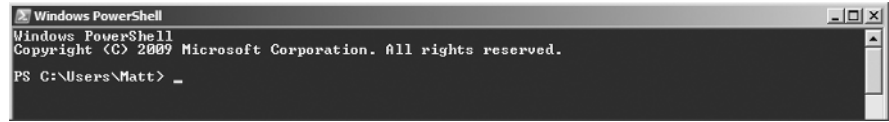
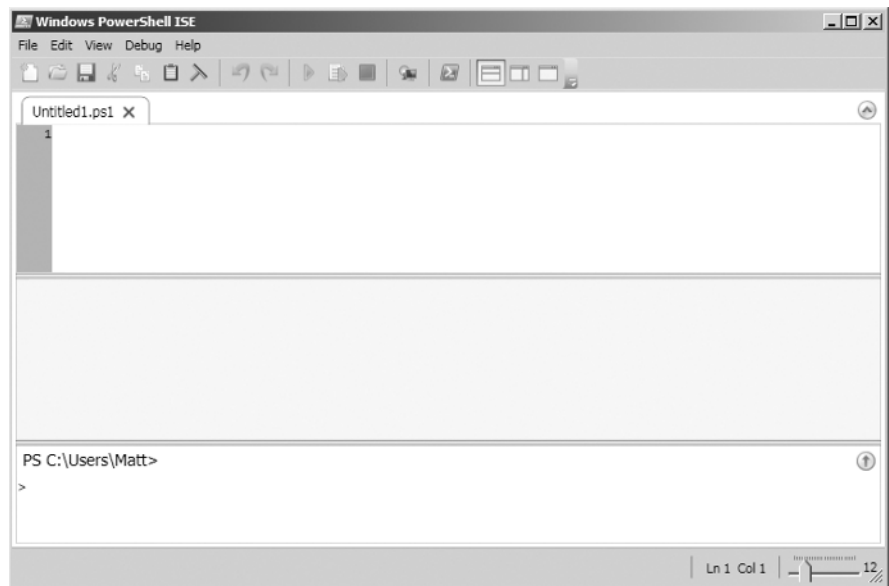


FIGURE 1.2 This is PowerShell too.



PowerShell can do that.

Once you learn the language, you should be able to perform these tasks faster than you have in the past.

By integrating PowerShell scripts into your environment, you can automate many of the time-consuming, monotonous tasks required of system administrators. If you look at tasks such as some of the previous examples that gather and parse large amounts of information, it may take a long time to do them manually. These types of tasks are perfect candidates for PowerShell scripting.

PowerShell includes numerous system administration utilities, consistent syntax and naming conventions, and improved navigation for common management data, such

as the registry, certificate store, and Windows Management Instrumentation (WMI). WMI is a core technology for Windows system administration, because it exposes a wide range of information in a uniform manner. PowerShell includes a cmdlet that allows you to interface with WMI objects, enhancing your ability to do real work.

But isn't PowerShell just a command-line tool? Yes, it is a command-line tool, but in most cases PowerShell can accomplish all the tasks that graphic management tools can.

PowerShell is built upon a robust architecture that includes the following:

- ▶ A script parser that processes language constructs, such as scripts, predicates, and conditionals
- ▶ A pipeline processor, which manages intercommand communication using pipes (|)
- ▶ A command processor, which manages command execution, registration, and associated metadata

In addition to those processors, the shell can also manage session state and has an extended type system, which exposes a common interface for accessing properties and methods independent of the underlying object type. Lastly, PowerShell includes a robust error handler for managing error exceptions and error reporting.

The Power Behind PowerShell

PowerShell is built around an object-oriented language that lets you manage your Windows infrastructure. It provides an interface and programming environment that allows users and administrators to access and set system properties through .NET objects and single-function command-line tools called cmdlets. Cmdlets are the building blocks for PowerShell scripts. Chapter 3 explores cmdlets and the core PowerShell syntax.

The scripting language manipulates objects (not text) using the .NET Framework and the .NET common language runtime. PowerShell is built on top of, and is integrated with, the Microsoft .NET Framework. It accepts and returns .NET objects, allowing for robust scripting that interfaces seamlessly with many line-of-business tools.

This is the main reason PowerShell is more than just a console application. It is a robust scripting environment that supports a full range of logical program control, including simple conditional statements and complex switch statements using regular expressions to parse conditions. Scripts can be used independently or in conjunction with other scripts, with .NET Framework or COM objects, or even in code. PowerShell enables easy access to COM and WMI to provide an environment for local and remote Windows systems.

In many cases, a majority of the built-in roles and services (such as IIS or Active Directory) that you may run on your Windows Server 2008 R2 server have PowerShell providers and cmdlets to manage them. For example, the PowerShell Provider for Internet Information Services (IIS) 7.5 allows you to easily automate routine and complex IIS 7.5 administration tasks, such as creating websites and managing configuration and runtime data by using PowerShell. Chapter 10 shows how to work with PowerShell and your websites.

All of the other major applications running on a Windows Server 2008 R2 server, including Microsoft Exchange Server, Microsoft SQL Server, and Microsoft SharePoint Server, have built-in support for PowerShell. (Exchange Server was the first major server application to get full support for PowerShell.) The SQL Server 2008 PowerShell snap-in supports more complex logic than Transact-SQL scripts, allowing SQL Server administrators to build robust administration scripts not only for server administration but also to extend the power of SQL databases. PowerShell in some cases is also replacing existing tools for the command prompt management of a server. With SharePoint Server, PowerShell is gradually replacing the `stsadm` tool, which has been the main tool for command prompt administration for SharePoint servers.

What About the Learning Curve?

One of the many benefits of PowerShell is that the learning curve to get started with it is minimal. If you already know scripting languages, you have a good base for working with PowerShell. Whether you have a background with command prompt tools for Microsoft or non-Microsoft operating systems such as UNIX, PowerShell lets you build on your existing command prompt knowledge. Throughout this book, you will see many examples of PowerShell that look similar to techniques you have used in other shells. PowerShell includes single-function tools such as `cd`, `copy`, and `dir` that you are familiar with from the Windows command interface. You can also recognize these other PowerShell functions from a UNIX background, such as `ls` or `man`.

If you have a UNIX administration background, you are familiar with the term *shell*. A shell provides a powerful, flexible, and scriptable command-line experience that allows you to perform any administrative task that you can perform using the console. The difference between using the shell and the PowerShell console is that the PowerShell is ideally suited to repetitive tasks. PowerShell is not a text-based shell but a console. PowerShell has a substantial number of built-in commands that provide you with a powerful tool set for script-based administration.

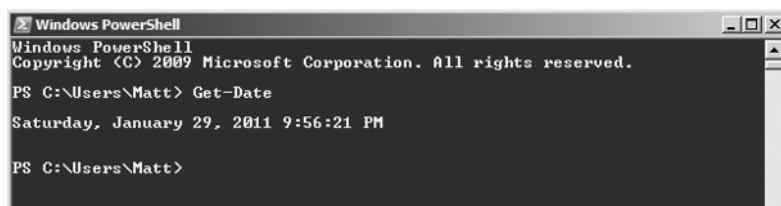
The formatting for commands that use the .NET Framework, COM objects, and WMI are slightly different from other scripting technologies, but in general those commands are simpler in PowerShell. If you are not familiar with scripting techniques, the base set of cmdlets is easy to learn, as you'll see throughout this book. PowerShell provides an intuitive scripting language specifically designed for day-to-day administrative of servers.

Cmdlets really showcase the intuitive nature of PowerShell. Cmdlets have a verb-noun structure, so they are somewhat self-describing. For example, here is a simple cmdlet that returns the current system date and time:

```
Get-Date
```

Your results will look similar to Figure 1.3.

FIGURE 1.3 A simple cmdlet



The cmdlets can also get more complex. In this book, you will start with the building blocks and get more in depth. Cmdlets can be used independently or scripted together to create a powerful automation application. Lastly, the language also provides a self-service help system, allowing you to learn the language quickly. Chapter 3 will show you how to get help by using the `Get-Help` cmdlet.

What's New in PowerShell 2.0?

With the launch of PowerShell 2.0, Microsoft began to take a deeper look into this language. With PowerShell being built into operating systems, IT administrators took notice. You may have been asking this question: "How can I leverage PowerShell in my environment, and where do I start?"

Microsoft wanted to make PowerShell 2.0 more enterprise-friendly so IT administrators everywhere could run, learn, and share PowerShell easily from within the GUI. PowerShell also had to be made to run safely and securely.

One of the new features in PowerShell 2.0 that allows IT administrators throughout the world to use PowerShell more easily is called *internationalization*. Internationalization enables PowerShell scripts to display messages in the language specified by the UI language setting on the user's computer. Under the hood, this feature queries the operating system of the user to determine what language is being used. This lets PowerShell display the appropriate language.

Microsoft added more than 50 cmdlets for the core PowerShell sessions. Although those new cmdlets are important, Microsoft also addressed many of the server roles on Windows Server that did not have native PowerShell cmdlets in PowerShell 1.0. One of the key roles on Windows Server 2008 R2 that got new cmdlets was Active Directory (AD). Managing AD with PowerShell 1.0 was a challenge. There were no built-in cmdlets, so you had to know how to work with LDAP in script. Chapter 8 takes a look at the new PowerShell cmdlets you can use to manage your AD environment. Chapter 8 will also show you a couple of new features — Recycle Bin and managed service accounts — you can manage only in PowerShell.

PowerShell in the Enterprise

PowerShell 2.0 provides several new capabilities to make the tool more enterprise-friendly. For example, running PowerShell commands on remote computers in PowerShell 1.0 was not built in. A lot of administrators started remote desktop sessions to run PowerShell commands. This was one of the challenges that was addressed in PowerShell 2.0.

Remoting uses the WS-Management protocol and the Windows Remote Management (WinRM) service that implements WS-Management in Windows. This protocol is a standard-based, firewall-compatible communications protocol. Chapter 6 covers remoting and shows you how to configure and work with PowerShell remoting.

Key to working with remoting is another new concept in PowerShell 2.0 called *sessions*. A session is the environment where you run PowerShell commands. Every time you start PowerShell, a new session is created. You can even create a new session in your existing session for a local or remote computer.

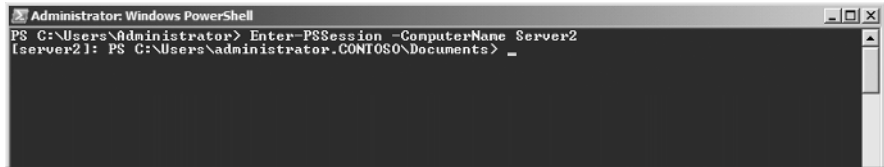
The session cmdlet uses a parameter called `ComputerName`. This allows you to specify the remote computer you want to start the PowerShell session on. For

example, the following cmdlet would create and enter a new PowerShell session on Server2:

```
Enter-PSSession -ComputerName Server2
```

Your results will look similar to Figure 1.4.

FIGURE 1.4 Remote session



Another key addition to PowerShell 2.0 is the ability to create and run background jobs. After you start a background job, you are returned almost immediately to your interactive PowerShell session. This allows you to continue to do work in your PowerShell session, and at any time you can see the status of your background jobs. The following command starts a command in the background to get the existing services:

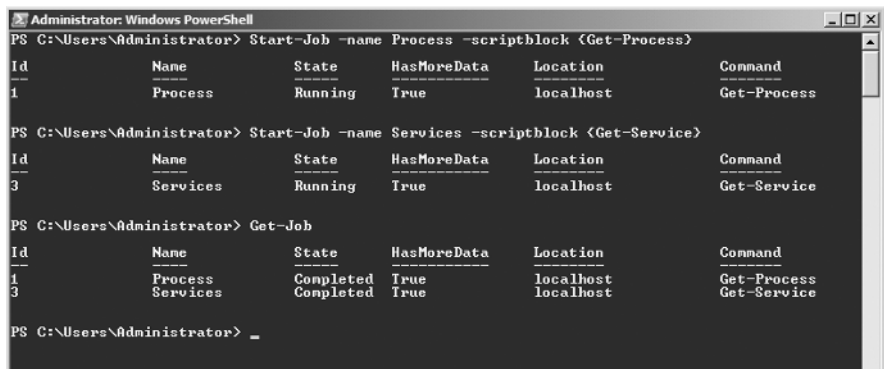
```
Start-Job -name Services -scriptblock (Get-Service)
```

To see the status of background jobs you started in your PowerShell session, you would run the following command:

```
Get-Job
```

Your results will look similar to Figure 1.5.

FIGURE 1.5 Background jobs



PowerShell with a GUI

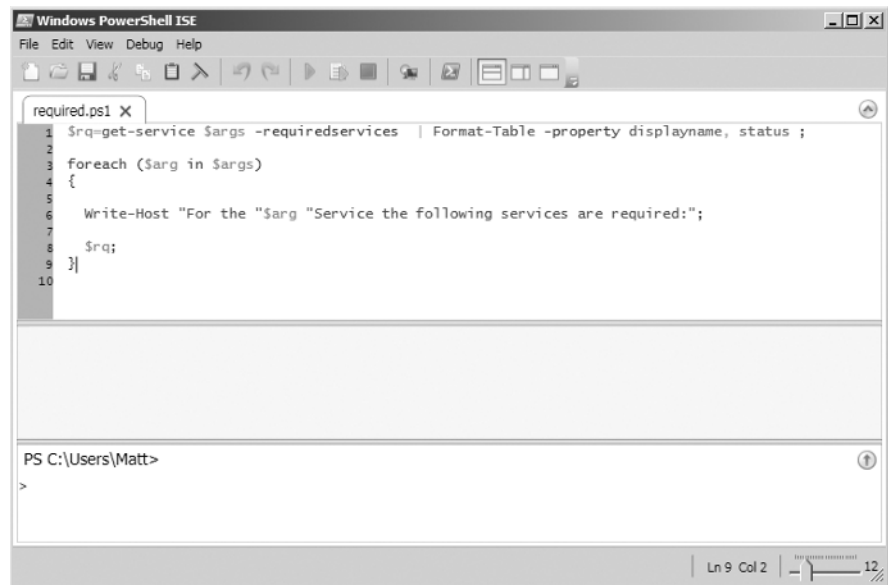
There was no built-in GUI in PowerShell 1.0, so you had only the command console for your PowerShell session. There were third-party tools you could use, such as PowerGUI (<http://powergui.org/index.jspa>).

With PowerShell 2.0, Microsoft added new features to take advantage of the GUI. The following are two of the main ways you can use PowerShell's GUI features:

- ▶ Integrated Scripting Environment (ISE)
- ▶ Out-GridView

The ISE shown in Figure 1.2 is a new GUI front-end application console for PowerShell. However, the primary benefit of the ISE is to create, edit, and debug PowerShell scripts. The ISE provides an easy-to-use, syntax-highlighted way to work with your scripts, as shown in Figure 1.6.

FIGURE 1.6 ISE with a script



There are debugging tools built in to PowerShell 2.0. Scripts are created in many different tools, from the ISE to Notepad, and scripters have used a variety of

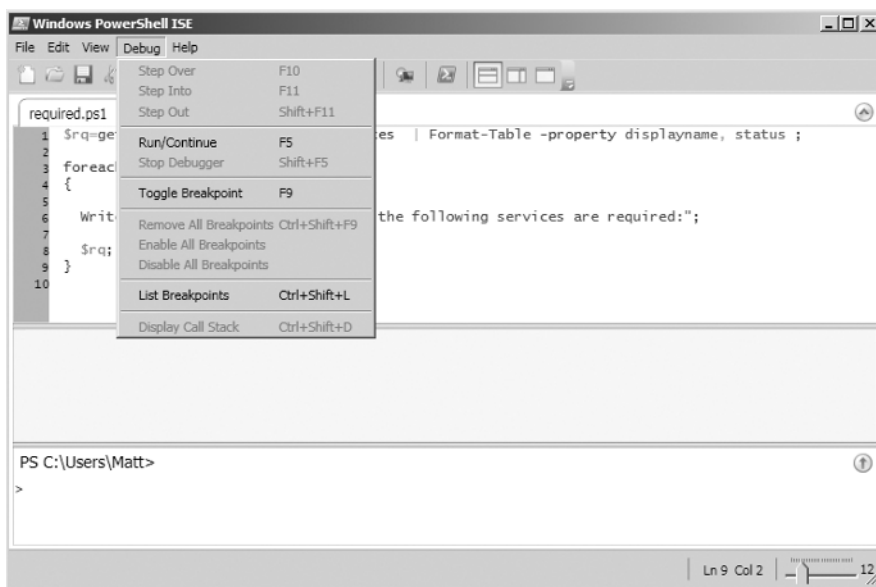
debugging tools with PowerShell. The new debugging features allow you set break-points on the following:

- ▶ Lines
- ▶ Columns
- ▶ Variables
- ▶ Commands

If you are using the debugger with your scripts, you can step into, over, and out of the scripts, and you can even display the call stack, often with a single keystroke. There are cmdlets to work with the debugger. You can also display the values of variables and run standard commands in the debugger.

The ISE makes it easy to interact with the debugger. Figure 1.7 shows the Debug menu.

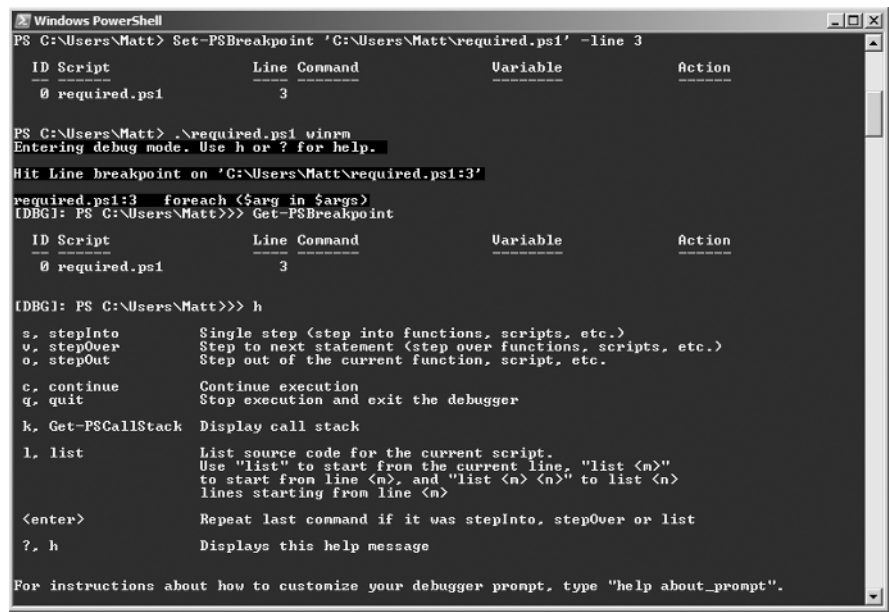
FIGURE 1.7 ISE's Debug menu



You can also access the debugger in your PowerShell sessions. You can set breakpoints using the `Set-PSBreakpoint` cmdlet, and you can list your breakpoints

with the `Get-PSBreakpoint` cmdlet for any of your PowerShell scripts. Figure 1.8 shows an example of a debugging session from the PowerShell console.

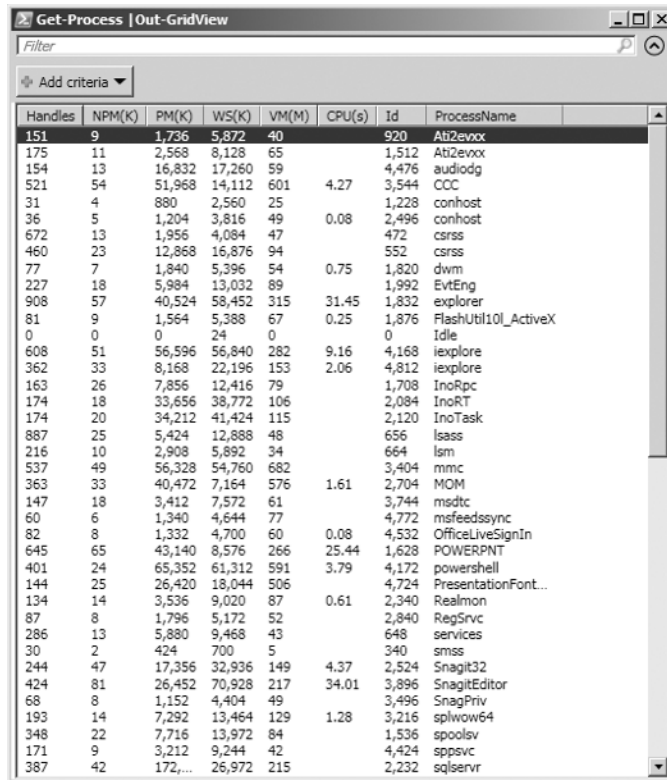
FIGURE 1.8 Debugging session in PowerShell



To learn more about the ISE, see Chapter 2 and Chapter 5. Chapter 2 shows you how to install the ISE, which may not be installed by default. Chapter 5 shows how to use this tool when working with PowerShell scripts.

Another way PowerShell leverages the Windows GUI is with the output cmdlet `Out-GridView`. This cmdlet allows you to take the output from a PowerShell command and display it in a Windows Explorer–style window, which not only displays your data but also allows you some interaction such as sorting and quickly filtering the data. For example, if you ran the command `Get-Process | Out-GridView`, your results would look similar to Figure 1.9.

You can click any of the column headings in the `Out-GridView` window, and the content will be sorted. You can also quickly filter the data by either adding criteria or typing in the Filter text box. Chapter 4 takes a look at the `Out-GridView` cmdlet as well as other ways to work with data from your PowerShell commands.

FIGURE 1.9 Out-GridView


Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
151	9	1,736	5,872	40		920	Ab2evvx
175	11	2,568	8,128	65		1,512	Ab2evvx
154	13	16,832	17,260	59		4,476	audiodg
521	54	51,968	14,112	601	4.27	3,544	CCC
31	4	880	2,560	25		1,228	conhost
36	5	1,204	3,816	49	0.08	2,496	conhost
672	13	1,956	4,084	47		472	csrss
460	23	12,868	16,876	94		552	csrss
77	7	1,840	5,396	54	0.75	1,820	dwm
227	18	5,984	13,032	89		1,992	EvtEng
908	57	40,524	58,452	315	31.45	1,832	explorer
81	9	1,564	5,388	67	0.25	1,876	FlashUtil10l_ActiveX
0	0	0	24	0		0	Idle
608	51	56,596	56,840	282	9.16	4,168	iexplore
362	33	8,168	22,196	153	2.06	4,812	iexplore
163	26	7,856	12,416	79		1,708	InoRpc
174	18	33,656	38,772	106		2,084	InoRT
174	20	34,212	41,424	115		2,120	InoTask
887	25	5,424	12,888	48		656	lsass
216	10	2,908	5,892	34		664	lsmon
537	49	56,328	54,760	682		3,404	mmc
363	33	40,472	7,164	576	1.61	2,704	MOM
147	18	3,412	7,572	61		3,744	msdtc
60	6	1,340	4,644	77		4,772	msfeedssync
82	8	1,332	4,700	60	0.08	4,532	OfficeLiveSignIn
645	65	43,140	8,576	266	25.44	1,628	POWERPNT
401	24	65,352	61,312	591	3.79	4,172	powershell
144	25	26,420	18,044	506		4,724	PresentationFont...
134	14	3,536	9,020	87	0.61	2,340	Realmon
87	8	1,796	5,172	52		2,840	RegSvc
286	13	5,880	9,468	43		648	services
30	2	424	700	5		340	smss
244	47	17,356	32,936	149	4.37	2,524	Snagit32
424	81	26,452	70,928	217	34.01	3,896	SnagitEditor
68	8	1,152	4,404	49		3,496	SnagitPriv
193	14	7,292	13,464	129	1.28	3,216	splwow64
348	22	7,716	13,972	84		1,536	spoolsv
171	9	3,212	9,244	42		4,424	sppsvc
387	42	172,...	26,972	215		2,232	sqlserv

PowerShell Has Something for Everyone

PowerShell has something for everyone, from IT professionals to developers to the casual scripter. PowerShell is a tool that can save you time and show you a new way to automate a task that was previously difficult or impossible. Unlocking PowerShell to meet your needs always starts with the basics.

Before you can dive into PowerShell to meet your particular interest or business, you need a solid foundation in PowerShell. You need to know the basics of installing PowerShell and of reading and writing PowerShell scripts. That way, you can build your knowledge for many other aspects of PowerShell. Whether your focus is IT administration or development, you need the basics.

PowerShell needs to be installed on any system you want to be able to manage with PowerShell. PowerShell can be installed on many Microsoft operating systems (including XP mode on Windows 7). There are third-party PowerShell add-ons for non-Microsoft operating systems. Knowing what systems you want to manage allows you to determine which is the best path to deploy PowerShell. Chapter 2 focuses on the proper way to enable and install PowerShell. PowerShell has a robust and easy-to-use built-in help system that provides descriptions of the various cmdlets, as well as examples in most cases.

After PowerShell has been installed, you can learn to read the language. When you see a command like the following, you should understand what it does:

```
Get-ADObject -SearchBase "CN=Deleted Objects,↵  
DC=your domain name,DC=Com" -Filter {lastKnownParent ↵  
-eq "OU=marketing,DC=deploy,dc=com"} -includeDeletedObjects↵  
| Restore-ADObject
```

This command restores deleted users from the marketing organizational unit (OU) in the `deploy.com` domain.

You can then begin to combine multiple commands into one script. You need to know how to shorten those commands and unlock many of the other administrative aspects of PowerShell. Working with scripts involves combining the tasks in the proper order and saving them in one file. There are websites that have PowerShell script repositories, and you can leverage the work of another PowerShell administrator. PowerShell also protects you from rogue PowerShell scripts and allows only those scripts that are safe and secure.

In Chapters 3–5, you will learn to master the basics. You will be able to break the previous command down into its smallest parts so commands like these do not scare you away from PowerShell. You will see how easy the language can be used to perform complex tasks.

What's in It for IT Professionals?

With Windows Server 2008 R2, you can install many roles and features to provide functionality to your infrastructure. From Active Directory to Hyper-V to IIS to Deployment Services, you can perform day-to-day administration with PowerShell.

After you learn the basics of the language, you need to put PowerShell in practice. When you install the features on your Windows Server 2008 R2 server, nearly all of

them have their own set of PowerShell commands and functions to perform a variety of tasks specific to the particular role or feature.

Beginning with basic installation of the roles and features on your server, PowerShell can be used to perform these functions for your full and core Windows Server 2008 R2 servers. Performing and scheduling a task such as a backup can be quickly created in a PowerShell script and tied to the Task Scheduler.

PowerShell can provide a consistent approach to the daily maintenance of servers. In some cases, PowerShell may be the only utility you can use. This is the case with the Active Directory Recycle Bin and managed service accounts, two features in Windows Server 2008 R2 Active Directory.

IIS provides another scenario for IT professionals to use PowerShell. With PowerShell, you can work with the core configuration to manage sites and work with web applications. This allows you to manage and quickly maintain web farms.

As an IT professional, you want PowerShell to be consistent when you work on various tasks or when you download third-party tools. This is where you see the pervasiveness of PowerShell. For example, when you download the Microsoft Deployment Toolkit (MDT), this free tool has built-in PowerShell cmdlets.

What makes PowerShell a unique tool set is the strong community following the language. In some cases, Microsoft did not provide cmdlets for a Windows Server 2008 R2 server role. Yet you can find third-party ones with an Internet search. This is the case with Hyper-V. With PowerShell 2.0, there are no built-in cmdlets to support working with Hyper-V, and you may have to use WMI to work directly with Hyper-V via PowerShell. However, the PowerShell community has created a dedicated provider for managing Hyper-V in PowerShell, making it easier than having to use WMI to accomplish the same tasks.

Chapters 7–12 focus on many of the daily workloads you may encounter when you manage a Windows Server 2008 R2 server with PowerShell. These chapters will show how to install server components; how to manage IIS, Hyper-V, and Active Directory; and how to use many other roles and features you will find in Windows Server 2008 R2.

What's in It for Developers?

Although this book does focus on some of the IT professional and administrative tasks performed on Windows Server 2008 R2 servers, there is a side of PowerShell

that developers can work with, making it that much more powerful and beneficial in your workplace. PowerShell is another development platform you can use to automate many tasks via code. After you have the foundational knowledge presented throughout this book, looking at the programmatic side of PowerShell will allow you to take PowerShell to another level.

PowerShell provides you with a lightweight (when compared to Visual Studio or other developer tools) programmatic interface. Many of the applications utilizing PowerShell have a core set of APIs accessible with PowerShell. There are features that were designed in PowerShell 2.0, such as transactions, geared to be used in code.

For those who are new to development and unfamiliar with the basic concepts of objects and properties, Appendix B explores objects and properties from a PowerShell perspective.

It may seem odd to develop on the command prompt. PowerShell has many tools to be able to extend the language into your developers' code. The ISE allows you to create full and robust scripts using PowerShell, with some familiar keyboard commands from Visual Studio. Not only does PowerShell have the tools, but it also has been designed to write your own advanced functions and cmdlets using programmatic logic and constructs at the command prompt. Appendix D covers working with advanced functions and cmdlets.

Being able to program with PowerShell allows you to create and work with your own providers. There are many providers built into your systems, but you may have a particular scenario where there is a gap and PowerShell does not have a tool set to help you. You can create your own custom providers, like the developers did for Hyper-V. These providers allow you to access data stored inside data stores such as the registry environment variables and certificate stores easier than former methods with a command line. Appendix C provides a guide for creating custom providers. PowerShell also provides the necessary tools and framework to be able to deploy the custom tools you create in your infrastructure. In PowerShell v1.0, these were called *snap-ins*; in PowerShell 2.0, *modules* make this even easier to do. Appendix E explains how to work with existing snap-ins and how to create your own.

Lastly, you can create GUIs in PowerShell. Whether you want to take advantage of Windows Presentation Foundation (WPF), with the separation of design and code, or continue the look of legacy applications with Windows Forms (WinForms), PowerShell allows you to work with both of these technologies. Although it is

relatively easy to work with GUI technologies in PowerShell, there are also some tools to make this easy process even easier. Appendix F explores creating GUIs in PowerShell.

EXERCISE 1: INVENTORY YOUR SCRIPTS

Take an inventory of the tasks you are currently using scripts to perform. By the end of the book you should be able to take the script or scripts you are currently using and convert them to PowerShell.

